



EMBEDDED  
LINUX  
CONFERENCE



THE LINUX FOUNDATION  
OPEN SOURCE SUMMIT  
NORTH AMERICA

# FROM UART TO PCIe and DMA: SELECTING CONNECTIVITY FOR YOUR FPGA-BASED SUBSYSTEM

Alexander Wirthmüller  
[aw@mpsitechnologies.com](mailto:aw@mpsitechnologies.com)

#ossummit



# Introduction

## About me

- Based in Munich
- Diploma in Electrical Engineering
- R&D Engineer at Mynaric (FPGA-based error-correction algorithms for free-space optical laser communications)
- Founder and Director at MPSI Technologies
- MPSI Technologies: make Embedded Software development more fun by replacing repetitive tasks by model-based source code generation

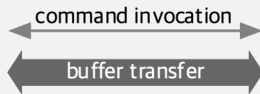


# Scenarios and interfaces

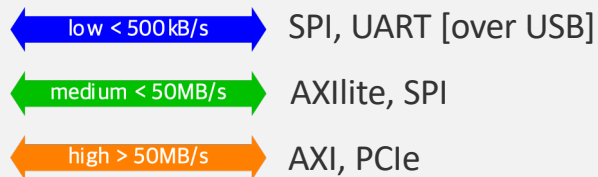
- FPGA module debugging
- Pre-processing / data reduction in FPGA
- FPGA result transfer to host
  - using DMA
- FPGA-SoC variant
  - with external peripherals
  - self-contained

## Legend

transaction purpose >



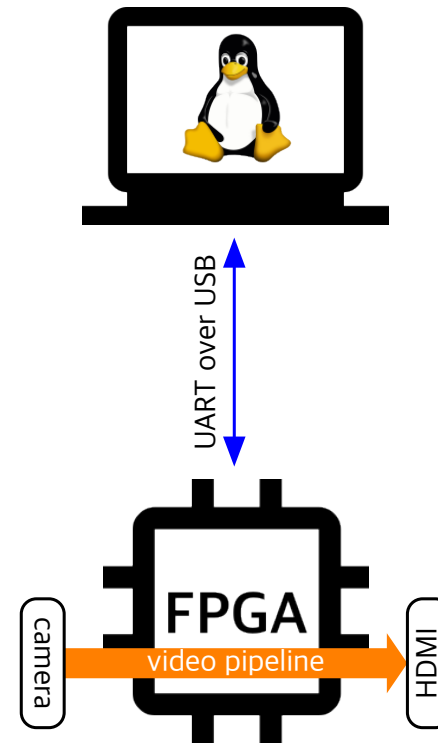
bandwidth >



SPI, UART [over USB]

AXI lite, SPI

AXI, PCIe

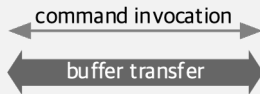


# Scenarios and interfaces

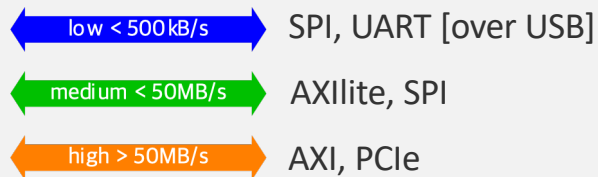
- FPGA module debugging
- Pre-processing / data reduction in FPGA
- FPGA result transfer to host
  - using DMA
- FPGA-SoC variant
  - with external peripherals
  - self-contained

## Legend

transaction purpose >



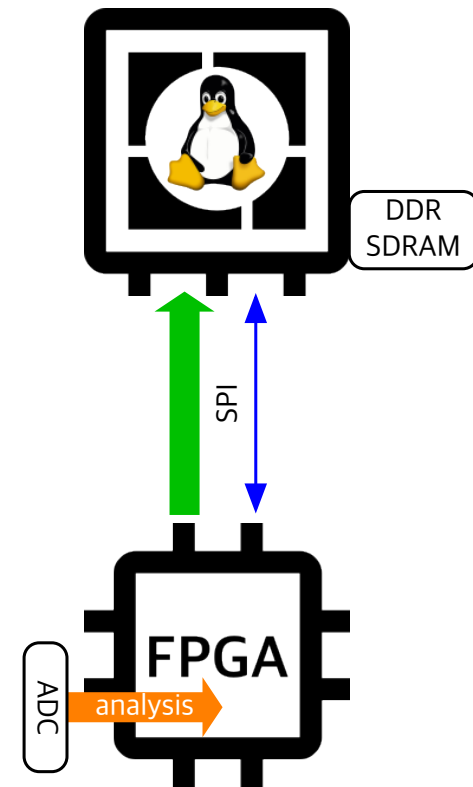
bandwidth >



SPI, UART [over USB]

AXI lite, SPI

AXI, PCIe

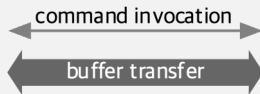


# Scenarios and interfaces

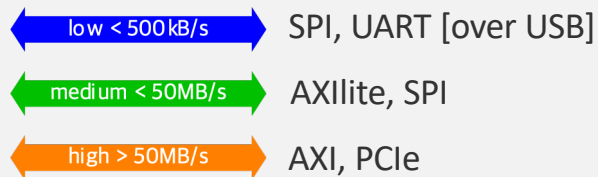
- FPGA module debugging
- Pre-processing / data reduction in FPGA
- FPGA result transfer to host
  - using DMA
- FPGA-SoC variant
  - with external peripherals
  - self-contained

## Legend

transaction purpose >



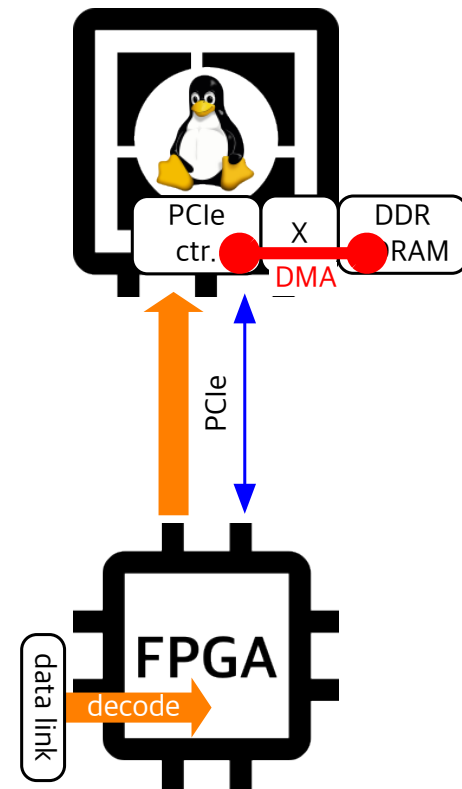
bandwidth >



SPI, UART [over USB]

AXI lite, SPI

AXI, PCIe

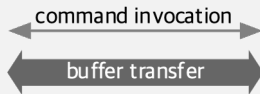


# Scenarios and interfaces

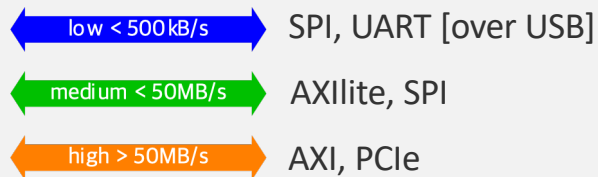
- FPGA module debugging
- Pre-processing / data reduction in FPGA
- FPGA result transfer to host
  - using DMA
- FPGA-SoC variant
  - with external peripherals
  - self-contained

## Legend

transaction purpose >



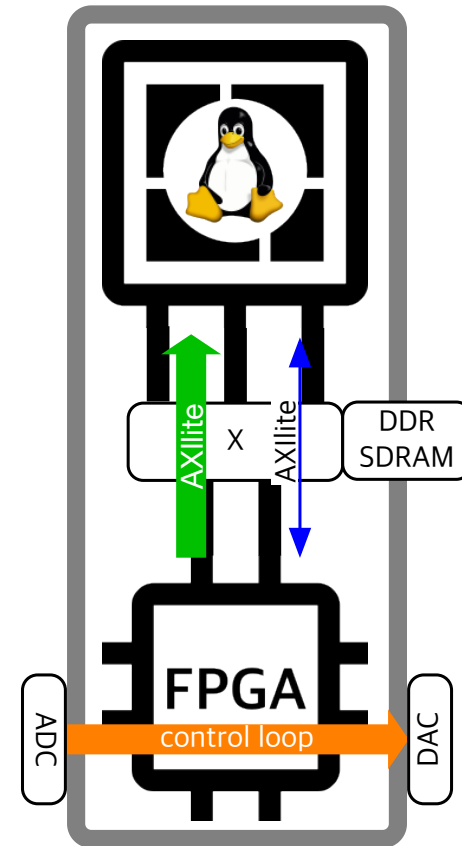
bandwidth >



SPI, UART [over USB]

AXI lite, SPI

AXI, PCIe

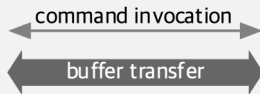


# Scenarios and interfaces

- FPGA module debugging
- Pre-processing / data reduction in FPGA
- FPGA result transfer to host
  - using DMA
- FPGA-SoC variant
  - with external peripherals
  - self-contained

## Legend

transaction purpose >



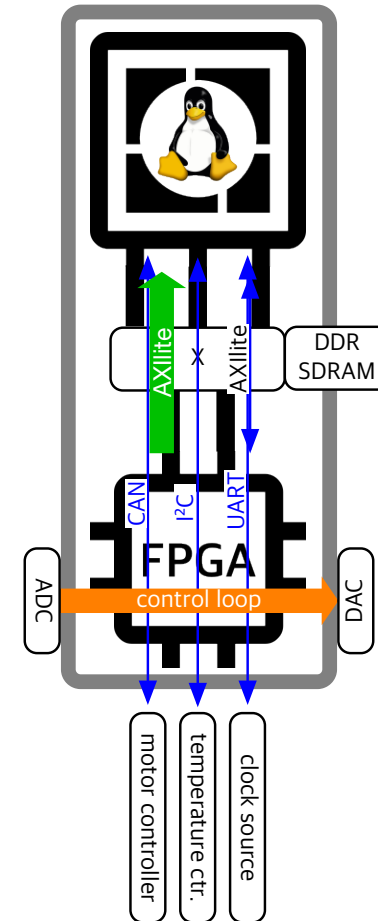
bandwidth >



SPI, UART [over USB]

AXI lite, SPI

AXI, PCIe

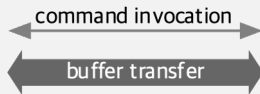


# Scenarios and interfaces

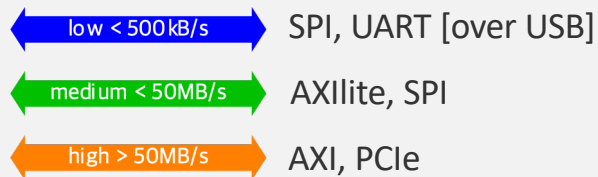
- FPGA module debugging
- Pre-processing / data reduction in FPGA
- FPGA result transfer to host
  - using DMA
- FPGA-SoC variant
  - with external peripherals
  - self-contained

## Legend

transaction purpose >



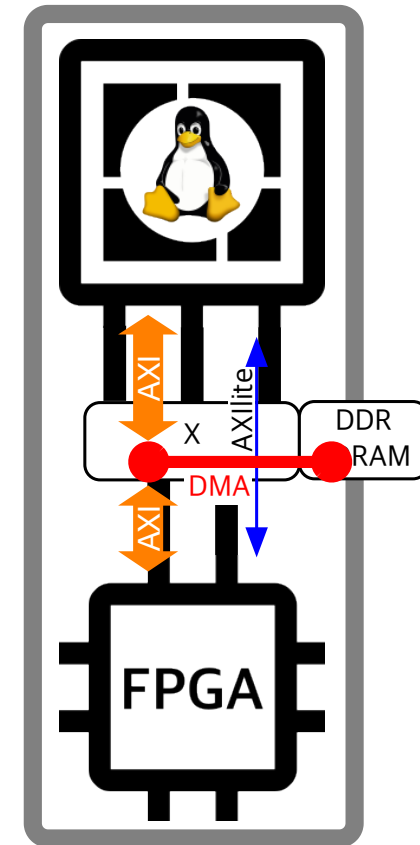
bandwidth >



SPI, UART [over USB]

AXI lite, SPI

AXI, PCIe

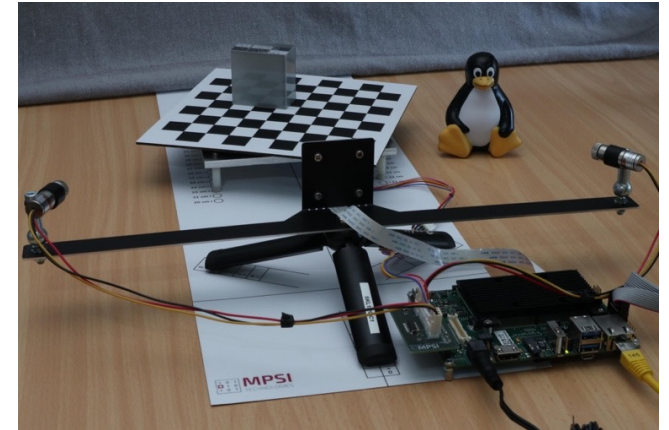




# Demo project: Tabletop 3D laser scanner

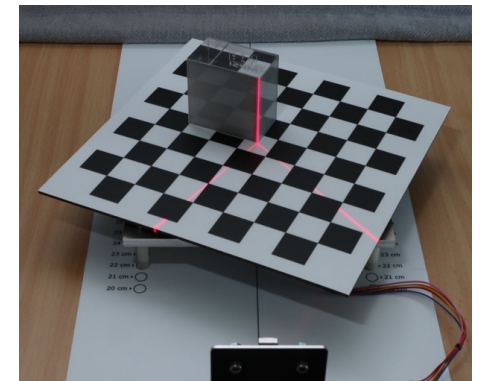
## Hardware variants | Key software functionality

- Lattice CrossLinkNX (volatile FPGA) evaluation board
  - a) UART-over-USB connected to laptop
  - b) PCIe connected to NXP i.MX6 (quad ARM) system
- Microchip PolarFire SoC Icicle kit (quad RISC-V + antifuse FPGA)
- Xilinx Zynq-7020 (dual ARM + volatile FPGA) developer board
- FPGA-less designs available, more FPGA-SoC designs under development



### Features

- Turntable with stepper motor
- Tripod with camera/laser holder
- IMX335 MIPI CSI-2 camera (5MP)  
max. data rate 150MB/s @30fps
- Two adjustable red line lasers

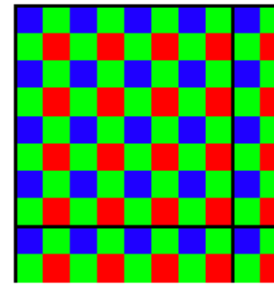


# Demo project: Tabletop 3D laser scanner

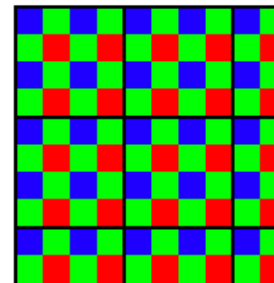
## Hardware variants | Key software functionality

- Preview image acquisition

2560 x 1920 -> 160 x 120 (color)



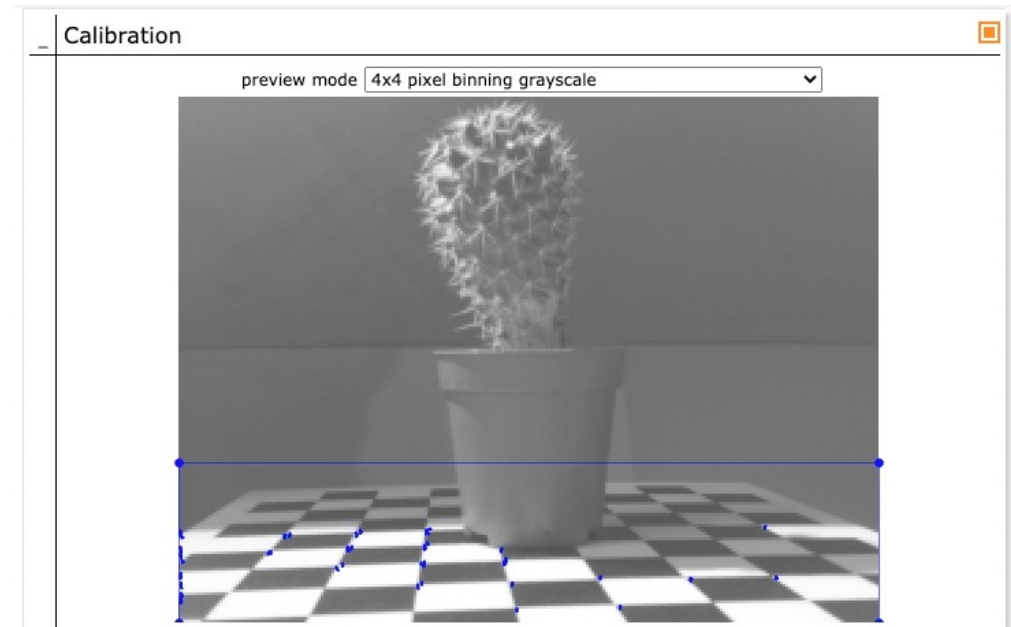
2048 x 1536 -> 512 x 384 (grayscale)



# Demo project: Tabletop 3D laser scanner

## Hardware variants | Key software functionality

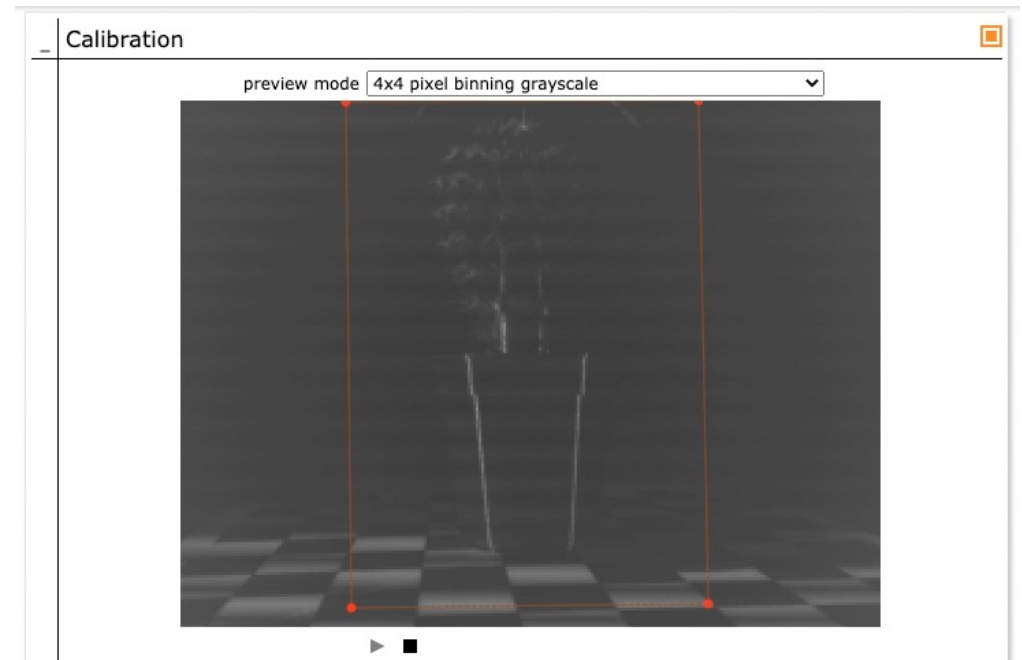
- Preview image acquisition
- Checkerboard corner detection for orientation



# Demo project: Tabletop 3D laser scanner

## Hardware variants | Key software functionality

- Preview image acquisition
- Checkerboard corner detection for orientation
- On/off identification of line laser traces in frames



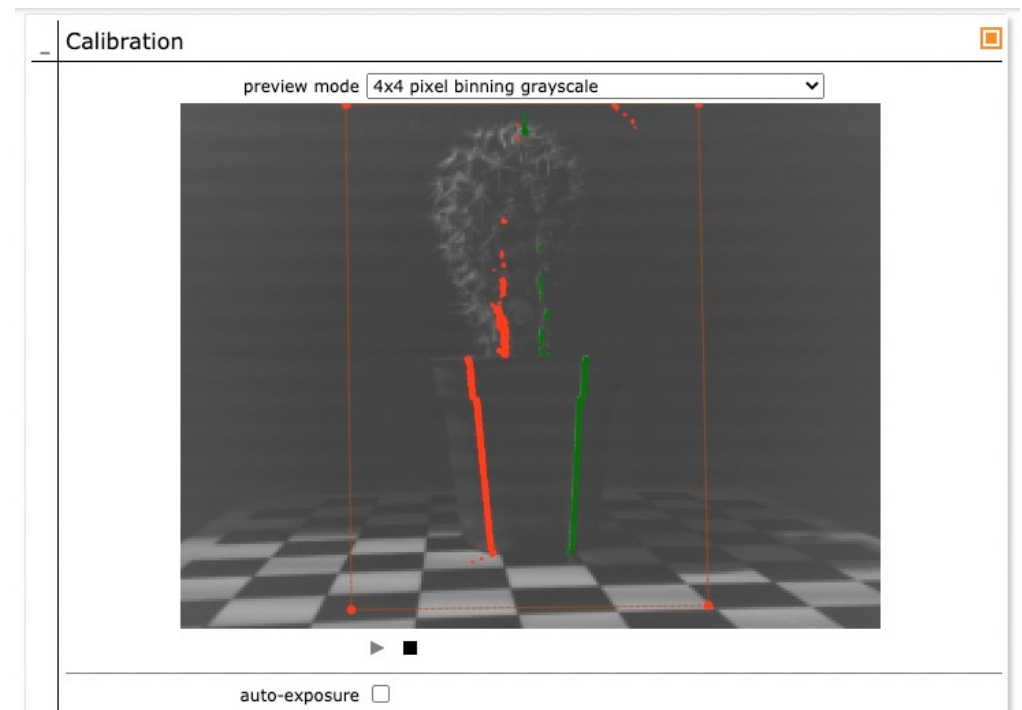
# Demo project: Tabletop 3D laser scanner

## Hardware variants | Key software functionality

- Preview image acquisition
- Checkerboard corner detection for orientation
- On/off identification of line laser traces in frames

each algorithm can be performed

→ either on the Linux host or on the FPGA, ←  
with varying load on the interconnect



# Outline

Bottom-up then full-circle

	<u>Linux host</u>	<u>FPGA</u>
application layer	C++ data processing	RTL algorithms, state machines, etc.
application layer handoff	target-specific C++ API library	target module RTL handshake
protocol layer	encode/decode C++ code	host interface RTL module
hardware abstraction layer	device driver	soft IP
physical layer	silicon IP and copper wires standard-compliant	

# Physical layer

	Linux host	FPGA
application layer	C++ data processing	RTL algorithms, state machines, etc.
application layer handoff	target-specific C++ API library	target module RTL handshake
protocol layer	encode/decode C++ code	host interface RTL module
hardware abstraction layer	device driver	soft IP
physical layer	silicon IP and copper wires standard-compliant	

	Complexity	Net bandwidth	Support	Conditions
UART	2 wire	400 kB/s	i.MX6 (all 32/64bit SoC's)	4 Mbps on-PCB routing
UART over USB	2 wire	417 kB/s	FTDI	x64 host USB2.0 hi-speed, FT232R
SPI	3 wire	4.8 MB/s	OMAP3xxx (all 32/64bit SoC's)	40 MHz on-PCB routing
AXI lite	(on-chip)	50 MB/s	Zynq (all FPGA-SoC's)	32 bit words, 100 MHz clock
PCIe	3 diff. pair, 4 wire	250 MB/s	CrosslinkNX (all mid-range FPGA's)	one lane PCIe 1.x, 2.5 Gbps
AXI4	(on-chip)	776 MB/s	PolarFire SoC (all FPGA-SoC's)	64 bit x 256 bursts, 100 MHz clock

# Hardware abstraction layer

## Device driver [host] | Soft IP [FPGA]

	Linux host	FPGA
application layer	C++ data processing	RTL algorithms, state machines, etc.
application layer handoff	target-specific C++ API library	target module RTL handshake
protocol layer	encode/decode C++ code	host interface RTL module
hardware abstraction layer	device driver	soft IP
physical layer	silicon IP and copper wires standard-compliant	

- Character device driver category
  - transfer control by host
  - from user space: `open()`, `ioctl()`, `read()`, `write()`, `close()`
  - UART:
    - typ. drivers: `bus/platform/drivers/imx-uart`
    - typ. device files: `/dev/ttyS*`, `/dev/ttymx*`
  - UART over USB:
    - typ. drivers: `bus/usb-serial/drivers/ftdi_sio`, `bus/usb-serial/drivers/cp210x`
    - typ. device files: `/dev/ttyUSB*`, `/dev/ttyACM*`
  - SPI:
    - typ. drivers: `bus/spi/drivers/spidev`
    - typ. device files: `/dev/spidev*.*`



# Hardware abstraction layer

## Device driver [host] | Soft IP [FPGA]

- Character device driver category (cont'd)
  - a simple cross-platform AXI lite driver, to appear at /dev/<pick\_your\_favorite>

```
static ssize_t device_read(
    struct file* file
    , char __user* buffer
    , size_t length
    , loff_t* offset
) {
    ssize_t retval = 0; // number of bytes read

    size_t i;
    u32 rdy;

    size_t len32 = length / 4; // align bytes to words
    if ((length % 4) > 0) len32++;

    void* buf = kmalloc(4 * len32, 0); // allocate memory

    if (buf) {
        for (i = 0; i <= 20; i++) { // wait for max. 210us
            if (i != 0) udelay(i);

            iowrite32(0xAAAAAAAA, mmAxilite + OFS_AXILITE_SETREAD); // REQ to read control address
            rdy = ioread32(mmAxilite + OFS_AXILITE_SETREAD);

            if (rdy == 0xAAAAAAAA) break; // ACK from read control address
        }
    }
}
```

```
if (rdy == 0xAAAAAAAA) {
    for (i = 0; i < len32; i++) // data from actual read address
        ((u32*) buf)[i] = cpu_to_be32(ioread32(mmAxilite + OFS_AXILITE_READ));

    iowrite32(0x55555555, mmAxilite + OFS_AXILITE_SETREAD); // ACK to read control address

    retval = length - copy_to_user((void __user*) buffer, buf, length);
};

kfree(buf);
};

return retval;
};
```

# Hardware abstraction layer

Device driver [host] | Soft IP [FPGA]

- Advanced category: PCIe and AXI4(-Burst)
  - typically, only makes sense combined with DMA and interrupts
  - the kernel's Userspace I/O has corresponding features, already integrated with PCIe  
<https://www.kernel.org/doc/html/v5.0/driver-api/uio-howto.html#making-the-driver-recognize-the-device>
  - helpful article by Oleg Kutkov regarding PCIe  
<https://olegkutkov.me/2021/01/07/writing-a-pci-device-driver-for-linux/>

# Hardware abstraction layer

Device driver [host] | Soft IP [FPGA]

- Generic RTL modules for UART and SPI

```
entity Uartrx_v1_1 is
  generic(
    fMclk: natural range 1 to 1000000;

    fSclk: natural range 100 to 50000000
  );
  port(
    reset: in std_logic;

    mclk: in std_logic;

    req: in std_logic;
    ack: out std_logic;
    dne: out std_logic;

    len: in std_logic_vector(16 downto 0);

    d: out std_logic_vector(7 downto 0);
    strbD: out std_logic;

    rxd: in std_logic;

    burst: in std_logic
  );
end Uartrx_v1_1;
```

```
entity Spislave_v1_0 is
  generic (
    cpol: std_logic := '0';
    cpha: std_logic := '0';

    nssByteNotXfer: std_logic := '0';
    misoPrecphaNotCpha: std_logic := '0'
  );
  port (
    reset: in std_logic;

    mclk: in std_logic;

    req: in std_logic;
    ack: out std_logic;
    dne: out std_logic;

    len: in std_logic_vector(16 downto 0);

    send: in std_logic_vector(7 downto 0);
    strbSend: out std_logic;

    recv: out std_logic_vector(7 downto 0);
    strbRecv: out std_logic;

    nss: in std_logic;
    sclk: in std_logic;
    mosi: in std_logic;
    miso: inout std_logic
  );
end Spislave_v1_0;
```

	Linux host	FPGA
application layer	C++ data processing	RTL algorithms, state machines, etc.
application layer handoff	target-specific C++ API library	target module RTL handshake
protocol layer	encode/decode C++ code	host interface RTL module
hardware abstraction layer	device driver	soft IP
physical layer	silicon IP and copper wires standard-compliant	

# Hardware abstraction layer

Device driver [host] | Soft IP [FPGA]

- Generic RTL code for AXI lite in two parts

```
process (extresetn, extclk)
-- ...
begin
    if extresetn='0' then
        stateOp <= stateOpInit;
        -- ...

    elsif rising_edge(extclk) then
        if stateOp=stateOpInit then
            -- ...
            stateOp <= stateOpIdle;

        elsif stateOp=stateOpIdle then
            if (axi_bvalid='1' and loc_waddr="00" and slv_reg0="AAAAAAAA") then -- host to FPGA begin request
                if rdyRx='1' then
                    enRx <= '1';
                end if;

                stateOp <= stateOpWrRdyA;

            elsif (axi_bvalid='1' and loc_waddr="10" and slv_reg2="AAAAAAAA") then -- FPGA to host begin request
                if rdyTx='1' then
                    enTx <= '1';
                end if;

                stateOp <= stateOpRdrdyA;
            end if;

            -- ...
        end if;
    end if;
end process;
```

```
entity Axirx_v2_0 is
port (
    reset: in std_logic;

    mclk: in std_logic;

    req: in std_logic;
    ack: out std_logic;
    dne: out std_logic;

    len: in std_logic_vector(21 downto 0); -- in words, max. 2^22-1

    d: out std_logic_vector(31 downto 0);
    strbD: out std_logic;

    rdyRx: out std_logic;
    enRx: in std_logic;

    rx: in std_logic_vector(31 downto 0);
    strbRx: in std_logic
);
end Axirx_v2_0;
```

# Hardware abstraction layer

Device driver [host] | Soft IP [FPGA]

- Vendor-specific IP for PCIe required
- Available from major vendors free of charge
- Survey mid-June 2022
  - intel PSG Cyclone-V: Avalon interfaces  
“Cyclone V Avalon {Memory Mapped/Streaming} Interface for PCIe Solutions” (UG-01110\_av{mm/st})
  - Lattice CrossLinkNX  
“PCIe X1 IP Core” (FPGA-IPUG-02091-1.4)
  - Microchip PolarFire SoC  
“PolarFire FPGA and PolarFire SoC FPGA PCI Express” (user guide)
  - Xilinx Zynq: AXI4-Stream interface  
“7 Series FPGAs Integrated for PCI Express v3.3” (PG054)

# Application and protocol layers

## Host to FPGA | FPGA to host

- Host: C++ API library forms byte code and initiates transfers guarded by CRC

	Linux host	FPGA
application layer	C++ data processing	RTL algorithms, state machines, etc.
application layer handoff	target-specific C++ API library	target module
protocol layer	encode/decode C++ code	host interface RTL module
hardware abstraction layer	device driver	soft IP
physical layer	silicon IP and copper wires standard-compliant	

Terminal

Connection state

Data in/out

```
tkclsrc.getTkst()
= (tkst=4680223)
tkclsrc.getTkst()
= (tkst=4711382)
step.moveto(angle=340,Tstep=150)
= ()
step.getInfo()
= (tixVState=move,angle=226)
step.getInfo()
= (tixVState=idle,angle=340)
```

Command execution

Command  Append

Command sequence

Submit

```
step.moveto(angle=340,Tstep=150)
tx 0x02 06 01 0005 78FA
1. 2. 3. 4. 5.
1. hostifToCmdinv
2. controller:step
3. command:moveto
4. length:5
5. CRC
tx 0x0154 96 7B65
1. 2. 3.
1. angle:340(uint16)
2. Tstep:150(uint8)
3. CRC
rx 0xAAAA
ACK
= ()

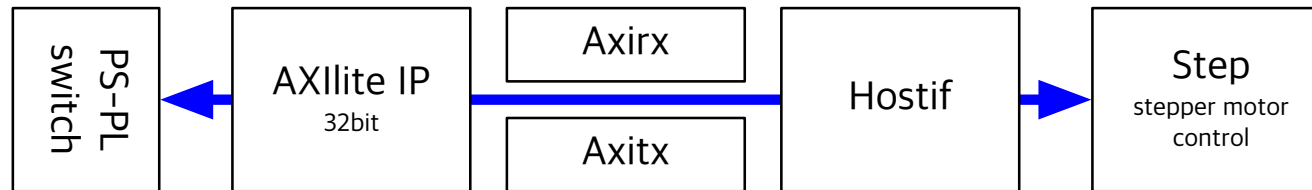
step.getInfo()
tx 0x01 06 00 0005 F865
1. 2. 3. 4. 5.
1. cmdretToHostif
2. controller:step
3. command:getInfo
4. length:5
5. CRC
rx 0x00 00E2 A8E6
1. 2. 3.
1. state:moving
2. angle:226(uint16)
3. CRC
=(tixVState=move,angle=226)

step.getInfo()
tx 0x01 06 00 0005 F865
rx 0x00 0154 AD52
=(tixVState=idle,angle=340)
```

# Application and protocol layers

## Host to FPGA | FPGA to host

- FPGA: "host interface" module decodes the byte string and triggers a handshake with the "step" target module



```
entity Step is
  generic (
    fMclk: natural range 1 to 1000000 := 50000 -- in kHz
  );
  port (
    reset: in std_logic;
    mclk: in std_logic;
    tkclk: in std_logic;

    ...

    reqInvMoveto: in std_logic;
    ackInvMoveto: out std_logic;

    movetoAngle: in std_logic_vector(15 downto 0);
    movetoTstep: in std_logic_vector(7 downto 0);

    ...

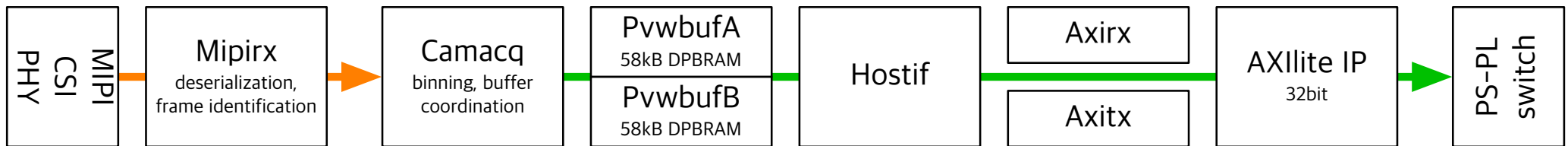
    nslp: out std_logic;
    m0: inout std_logic;
    dir: out std_logic;
    step0: out std_logic
  );
end Step;
```

# Application and protocol layers

## Host to FPGA | FPGA to host

	Linux host	FPGA
application layer	C++ data processing	RTL algorithms, state machines, etc.
application layer handoff	target-specific C++ API library	target module
protocol layer	encode/decode C++ code	host interface RTL module
hardware abstraction layer	device driver	soft IP
physical layer	silicon IP and copper wires standard-compliant	

- FPGA: reduce 2560x1920 YUV images @30fps (150MB/s) to 160x120 RGB images (1.73MB/s), then provide to host in A/B buffer



- Host: poll the buffer status, initiate buffer transfer and display

```
while (true) {
    if (shrdat.cancelPvw) break;

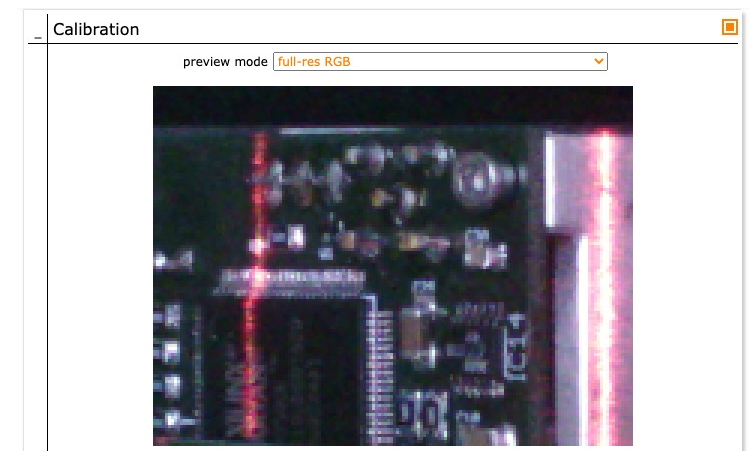
    shrdat.mPvw.lock("JobWzskAcqFpgapvw", "runPvw[2]");

    srv->srcarty->camacq_getPvwinfo(tixVPvwbufstate, tkst);

    if ((tixVPvwbufstate == VecVWskdArtyCamacqPvwbufstate::ABUF) || (tixVPvwbufstate == VecVWskdArtyCamacqPvwbufstate::BBUF)) {
        if (tixVPvwbufstate == VecVWskdArtyCamacqPvwbufstate::ABUF) srv->srcarty->shrdat.hw.readPvwbufFromCamacq(sizeBuf, buf, datalen);
        else if (tixVPvwbufstate == VecVWskdArtyCamacqPvwbufstate::BBUF) srv->srcarty->shrdat.hw.readPvwbufFromCamacq(sizeBuf, buf, datalen);

        shrdat.mPvw.unlock("JobWzskAcqFpgapvw", "runPvw[2]");
    } else {
        shrdat.mPvw.unlock("JobWzskAcqFpgapvw", "runPvw[3]");

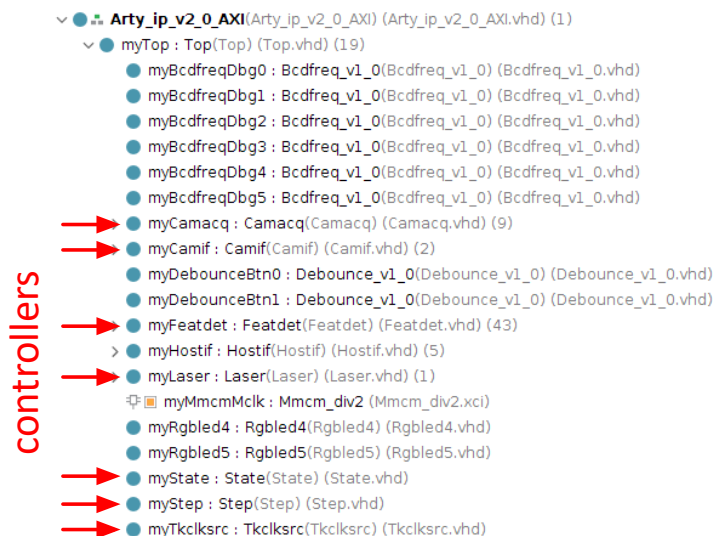
        nanosleep(&deltat, NULL);
    }
};
```





## Less repetitive coding and consistent results

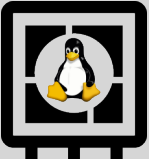

- “Single source of truth” generation of code for host and FPGA sides
  - One model file for modular structure, specifying “controllers” aka. RTL modules with commands / buffer transfers attributed
  - Another model file to specify those commands and buffer transfers



arty_ip_AXI;top	step								
	ImelMController.								
	^								
		ImelMVector2.sr	sref	srefsKOption					
		tixlin	VecVWskd	filfed;notit					
			ImelMVect	Title	Comment				
			idle						
			move						
			ImelMVectoritem2.end						
		ImelMVector2.end							
		ImelMCommand.sref	sref	srefixVRettye	sreflvRefWdbeMSignal	srefRvr	srefRerRef	Comment	
		0	getInfo	statsng					
			ImelAMCo	srefixWdbeVPart	srefRefWdbeMVector	Length	Comment		
			tixVState	tix	VecVWskdArtyStepState				
			angle	uint16				in stepper motor steps (1000 per rev.)	
			ImelAMCommandRetpar2.end						
		0	moveto	void					
			ImelAMCo	srefixWdbeVPart	srefRefWdbeMVector	Length	Defval	srefRefWdbe	Comment
			angle	uint16			0		in stepper motor steps (1000 per rev.)
			Tstep	uint8			60		in tkclk clocks: rps = 10000 / (Tstep * 200 * 5)
			ImelAMCommandInvar2.end						
		0	set	void					
			ImelAMCo	srefixWdbeVPart	srefRefWdbeMVector	Length	Defval	srefRefWdbe	Comment
			rng	_bool			fis8		
			ccwNotCw	_bool			fis8		
			Tstep	uint8			60		in tkclk clocks: rps = 10000 / (Tstep * 200 * 5)
			ImelAMCommandInvar2.end						
		0	zero	void					
		ImelMCommand2.end							
	ImelMController.end								

# Three FPGA API concepts

Simple, callback-based, register-based

	Simple (cf. slides before)	Callback-based	Register-based
	blocking access to FPGA resource for all command / buffer transfer requests	commands / buffer transfers can be launched (non-blocking), with notification on reply	no commands / buffer transfers exist; targets and their {set/get}table parameters are written / polled
	one-at-a-time processing in host interface	command invocation / return buffer FIFO's	simple mapping
Pros	<ul style="list-style-type: none"><li>• low FPGA footprint</li></ul>	<ul style="list-style-type: none"><li>• delayed and multiple command returns possible</li></ul>	<ul style="list-style-type: none"><li>• industry standard, good also for bare-metal</li></ul>
Cons	<ul style="list-style-type: none"><li>• host may have to poll status from multiple threads</li></ul>	<ul style="list-style-type: none"><li>• larger FPGA footprint</li></ul>	<ul style="list-style-type: none"><li>• lack of comfort, prone to error</li><li>• undefined behavior as update order not guaranteed</li></ul>

# Conclusion

- Linux host to FPGA connectivity is not rocket science
- What helps
  - Few, established hardware interfaces are used throughout the industry
  - The corresponding FPGA IP is free, ideally Open Source
  - On the Linux side, there is convenient character device and UIO driver support
- Model-based generation of source code for both sides simplifies life further
  - Designs become interface-agnostic
  - Vendor lock-in can be avoided

# Thank You!

## Questions?

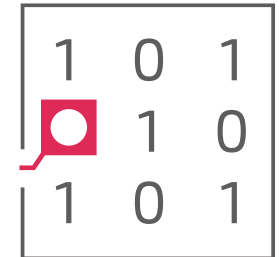
Also, feel free to connect.

- <https://www.linkedin.com/in/wirthmua>
- <https://github.com/mpsitech>

Alexander Wirthmüller  
Founder & Director

Phone: +49 (89) 4524 3826  
Mobile: +49 (175) 918 5480  
E-Mail: [aw@mpsitech.com](mailto:aw@mpsitech.com)

**MPSI** Technologies GmbH  
Agnes-Pockels-Bogen 1  
80992 Munich  
Germany  
[www.mpsitech.com](http://www.mpsitech.com)



**MPSI**  
TECHNOLOGIES



# EMBEDDED LINUX CONFERENCE



OPEN SOURCE SUMMIT  
NORTH AMERICA

THE LINUX FOUNDATION

